REPORT DOCUMENTATION PAGE

Form Approved
OMB NO. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 4th 2007	Final Oct 1, 2004 - Sept 30, 2006

4. TITLE AND SUBTITLE
DEVELOPING COLLABORATIVE PROFILES OF ATTACKERS: A LONGITUDINAL STUDY

5. FUNDING NUMBERS
W911NF-04-1-0442

6. AUTHOR(S)
Salvatore Stolfo(PI), Janak Parekh(GRA), Michael Locasto(GRA)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
COLUMBIA UNIVERSITY, COMPUTER SCIENCE DEPARTMENT, 450 CSB, NY, NY 10027

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U. S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
47544.7-CI

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.

12 a. DISTRIBUTION / AVAILABILITY STATEMENT	12 b. DISTRIBUTION CODE
Approved for public release; distribution unlimited.	167AA

13. ABSTRACT (Maximum 200 words)

We implemented a new content anomaly detector, Anagram, which models a mixture of high-order n-grams ($n > 1$) designed to detect anomalous and "suspicious" network packet payloads. For both Anagram and previously developed anomaly detector, Payl, we explored possible ways in which payload-based correlation can be applied, so that the alerts generated by both sensors can be included in our "collaborative security" infrastructure, called Worminator. Worminator is designed to exchange information securely, privately and in real-time between sites in order to reveal an accurate view of external threats, especially stealthy ones. To address the need for efficient alert correlation, we introduced the notion of network scheduling: the controllable formation and dissolution of relationships between nodes and groups of nodes in a network. Our network scheduling mechanism is a procedure for coordinating the exchange of information between the members of a correlation group. The mechanism is controlled by a dynamic and parameterizable correlation schedule. We performed a longitudinal study which is designed to demonstrate the proposed Worminator hypothesis, that collaborative intrusion detection not only enables detection of worm spread but also scanning behavior as precursors to an attack. There are three key longitudes for analysis: over time, over geographical and network space and by target.

14. SUBJECT TERMS	15. NUMBER OF PAGES
DISTRIBUTED INTRUSION DETECTION, COLLABORATIVE SECURITY, WORMINATOR	
	16. PRICE CODE

17. SECURITY CLASSIFICATION OR REPORT	18. SECURITY CLASSIFICATION ON THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	**UNCLASSIFIED**	**UNCLASSIFIED**	**UL**

NSN 7540-01-280-5500

Standard Form 298 (Rev.2-89)
Prescribed by ANSI Std. 239-18
298-102

FINAL PROGRESS REPORT
Developing Collaborative Profiles of Attackers: A Longitudinal Study

1. ContentThe Final Progress Report covers the entire performance period. Please do not delay submission while you are waiting for Reprints of publications.
2. Requirement: Final Progress Reports must include:
    a. A "Memorandum of Transmittal", Enclosure 3.
    b. A Filled out DD Form 882 (Report of Inventions and Subcontracts)
    c. A "Final Progress Report", including the following information:
        (1) Foreword (optional)
        (2) Table of Contents (if report is more than 10 pages)
        (3) List of Appendixes, Illustrations and Tables (if applicable)
        (4) Statement of the problem studied
        (5) Summary of the most important results
        (6) Listing of all publications and technical reports supported under this grant or contract. Provide the list with the following breakout, and in standard format showing authors, title, journal, issue, and date.
            (a) Papers published in peer-reviewed journals: NONE
            (b) Papers published in non-peer-reviewed journals or in conference proceedings

M. Locasto, J. Parekh, A. Keromytis, S. Stolfo "Towards Collaborative Security and P2P Intrusion Detection", Proceedings of the 2005 IEEE Workshop on Information Assurance and Security, June 2005

Ke Wang, Janak J. Parekh, Salvatore J. Stolfo "Anagram: A Content Anomaly Detector Resistant To Mimicry Attack" In Proceedings of the Nineth International Symposium on Recent Advances in Intrusion Detection(RAID 2006)

Janak J. Parekh, Ke Wang, Salvatore J. Stolfo " Privacy-Preserving Payload-Based Correlation for Accurate Malicious Traffic Detection " In SIGCOMM Workshop on Large Scale Attack Defence 2006

Ke Wang, "Network Payload-based Anomaly Detection and Content-based Alert Correlation", PhD thesis, 2006

Janak J. Parekh, "Privacy-Preserving Distributed Event Corroboration", PhD Thesis, 2007

            (c) Papers presented at meetings, but not published in conference proceedings

M. Locasto, J. Parekh, S. Stolfo, A. Keromytis, T. Malkin, V. Misra "Collaborative Distributed Intrusion Detection", CU Tech Report CUCS-012-04, 2004

CUCS D-NAD Group "On the Feasability of Distributed Intrusion Detection", Technical Report, Sept 2004

            (d) Manuscripts submitted, but not published: NONE
            (e) Technical reports submitted to ARO
        (7) List of all participating scientific personnel showing any advanced degrees earned by them while employed on the project

Janak Parekh

        (8) Report of Inventions (by title only)

Systems. Methods, and Media for Outputing Data Based on Anomaly Detection, 2006

        (9) Bibliography
        (10) Appendixes
    d. A "Standard Form 298 (Enclosure 1)", including the following required entries:
        (1) Block 2, Report Date
        (2) Block 3, Report Type and Dates Covered
        (3) Block 4, Proposal Title
        (4) Block 5, Contract/Grant Number
        (5) Block 6, Author(s)
        (6) Block 7, Performing Organization Name(s) and Address(es)

(7) Block 13, Abstract (must not exceed the 200 word limitation)
(8) Block 14, Subject Terms
(9) Block 15, Number of Pages

1 Payload Anomaly Detection and Alert Correlation

1.1 Payload Anomaly Detection

We have developed two methodologies to analyze and model normal payloads that are expected to be delivered to network services or applications: PAYL, which implements anomaly detection based on frequency-based 1-gram modeling, and Anagram, which uses binary-based mixtures of higher order n-gram modeling ($n > 1$). Both sensors train on *normal* unencrypted content flows and employ service-specific models to test for suspicious traffic.[1] Alerts are generated on traffic sufficiently deviant from normal; it is these alerts that we wish to share with other sites to resolve false positives from true zero-day attacks.

PAYL: 1-gram frequency modeling PAYL's models are 1-gram byte frequency distributions conditioned on packet length; tested traffic is classified as normal or malicious by computing the Mahalanobis distance between the distribution of the candidate packets and the frequency model. A larger distance means bigger deviation from the model and a more abnormal packet; thresholding differentiates normal from malicious traffic.

A *raw* PAYL alert typically contains metadata, including the source and target IP/port pair, payload length, and score (distance from model). Additionally, the suspicious packet may be included in its alert. While the payloads can be shared, they significantly increase alert sizes and run into privacy issues, especially for misclassified traffic, i.e. false positives. While PAYL's false positive rates have been determined to be very low [11], the notion of transmitting any raw payload inhibits collaboration among defensive sites.

Anagram: n-gram binary modeling Anagram uses an alternative approach to anomaly detection via *binary-based high order n-gram modeling*. Compared to 1-gram, higher order n-grams are better at modeling sequential content information in packets, and thus it is capable of detecting significant anomalous byte sequences and their location within a packet. To avoid significant memory overhead associated with n-gram frequency distributions, only a binary (yes/no) statistic is kept for each possible gram. Scoring is accomplished by counting the percentage of not-seen-before (i.e. unusual) n-grams out of the total n-grams in the packet, and thresholding is again applied to differentiate traffic.

Surprisingly, analysis shows [12] that binary-based modeling produces extremely good results; it turns out the additional data representation of frequency-based modeling is less advantageous when the space of potential grams grows significantly (e.g., the likelihood of having significant frequency information for distinct 5-grams, or 256^5 grams, is significantly smaller than for the 256 distinct 1-gram), and the representational power of higher-order n-grams effectively offsets the loss of frequency information.

The structure of a raw Anagram alert is similar to that of a raw PAYL alert.

Bloom filters Even though binary-based modeling significantly reduces space overhead, there is still a significant number of possible n-grams as n increases, and a typical hash set structure uses at least 4 bytes per entry. Since only the binary set property is needed, we can use a more efficient, bit-based representation to store the model, reducing data requirements by an order of magnitude. A *Bloom filter* [1] is one such structure; it is represented as a bit array of n bits, where any individual bit i is set if the hash of an input value, $\texttt{mod}\ n$, is i.

A Bloom filter contains no false negatives, but may contain false positives if collisions occur; the false positive rate can be optimized by changing the size of the bit array to avoid saturation, as well as using multiple hash functions (and requiring all of them to be set for an item to be verified as present in the Bloom filter). Operations on a Bloom filter are also $O(1)$, keeping computational overhead low. Finally, a Bloom filter has interesting privacy-preserving properties; we explore these in the next section.

[1] Anagram utilizes other information and is semi-supervised.

1.2 Correlation Techniques

We developed several techniques (both raw and privacy-preserving) to support content-based alert correlation. First, however, we develop several metrics as to how we can best compare these techniques.

Evaluating correlation techniques On one extreme, we can consider the idea of transmitting the raw packets that generated alerts; while this enables any correlation technique, we consider it infeasible because of the sheer amount of data and the fact it is not privacy-preserving. On the other end of the spectrum, we can consider privately-encrypted packet content: unless the key is shared, it essentially appears as noise to peers—but this requires all or no trust. The techniques in this paper fall somewhere in between, and we characterize their relative merits from two perspectives: our ability to correlate data given a transformed version of packets and the amount of privacy that is gained using different privacy-preserving transformations of packet content.

Correlation ability. The fundamental question, given any technique, is whether it is possible to correlate alerts with low false positive and low false negative rates. Given raw packets that generate an alert, there are several well-defined algorithms that aim to accomplish this task. We consider the *longest common subsequence*, or LCSeq, as an appropriate baseline, as it is able to find any non-semantic commonality in the candidate packets, and discuss it below. Other approaches are outside the scope of this paper, which focuses on correlation amongst pure network sensors, i.e. no host-specific information.

Given a technique, and a collection of alerts, we can then compute a *similarity score distribution* as each pair of alerts is tested (see section 1.3). This score distribution then becomes a useful metric for comparing correlation ability. If we consider LCSeq as a useful baseline, for instance, we can measure the deviation of other techniques from LCSeq as a comparative measure of how other techniques correlate alerts. Ideally, a network sensor would be able to use a privacy-enabled technique and get similar results, signifying an increase in the privacy preservation while maintaining the ability to determine common threats and exploits.

Correlation speed. Finally, one remaining important characteristic is the ability to correlate *quickly*, especially if many sites are involved with many alerts being generated and exchanged. This "speed" metric is reflected in two aspects: the resulting alert size after a transformation is applied, and the computation overhead necessary to transform the original alert. As with the previous cases, we consider raw packets the baseline: it is the largest unencrypted alert encoding (up to 1500 bytes, i.e. bounded by packet size, per alert) and LCSeq is amongst the slowest correlation mechanisms (up to polynomial-time with respect to buffer size).

Alert correlation We correlate content alerts using three main approaches: raw packet alert correlation, frequency-based alert correlation, and n-gram alert correlation. Techniques for other alerts (e.g., IP alerts) are considered outside the scope of this paper.

Baseline: Raw payload correlation

As previously discussed, we choose raw packet alert correlation as a baseline technique: it contains the most complete original information.

SE: String Equality. This is the simplest and most intuitive correlation approach. Two alerts are deemed similar to each other only if they have identical content. This metric is very strict and does minimize false positives, but has no tolerance for any variation—fragmentation, polymorphism, obfuscation, etc. Equality is memory and computationally efficient (linear time).

LCS: Longest Common Substring. LCS is one of the classic string comparison techniques; it is less deterministic than SE, and is not susceptible to fragmentation. The longer the string that LCS computes, the greater the confidence that the compared alerts are similar. While it allows minor payload manipulation, multiple changes often cause a short LCS, reducing confidence in its correlation ability. LCS is reasonably fast; a suffix-tree implementation is linear-time, but at the cost of having to store a suffix tree per alert (or $O(n^2)$ for a naive but memory-efficient algorithm).

LCSeq: Longest Common Subsequence. LCSeq can be considered a generalization of LCS; instead of finding a single contiguous matching block, LCSeq allows non-matching characters to be interposed. This enables detection despite a variety of payload manipulation operations, including insertion and reordering, and potentially polymorphism. Like LCS, the length of a LCSeq is an indication of similarity. Its main shortcoming is its computation overhead; at best, sparse dynamic programming can achieve, on average, $O(n \lg n)$ complexity (and can range to $O(n^2 \lg n)$ worst-case).

ED: Edit Distance. Edit distance, also known as Levenshtein distance, is another commonly-used approach to compare string similarity. It computes the smallest number of insertions, deletions, and substitutions required to change one string into another. In general, it has similar properties as LCSeq.

Frequency-modeled 1-gram alert correlation

Having discussed different techniques for raw payload comparison and correlation, we now describe our first alert transformation: frequency modeling. As our work on PAYL demonstrates [13], 1-gram frequency models are a good indicator of the nature of packet content. We can leverage this technique and use frequency distributions as alerts, either with the corresponding normalized frequency counts or with an approximation of this information.

Frequency Distribution. A packet payload can be represented by its byte frequency distribution, making it nearly impossible to reconstruct the actual payload except in degenerate cases—the byte distribution contains byte values but no sequential information. Given two packets with their respective distributions, we can apply standard distance metrics to determine similarity; Manhattan distance is efficient ($O(n)$ in length of the alert) yet produces a good approximation of the actual distance. Frequency-based alerts are comparatively sized compared to packets; a floating-precision frequency distribution takes 1KB of space.

Z-String. A more compact frequency representation based upon the packet payload's byte distribution is what we term a "Z-String", short for "Zipf String" [13]. As its name implies, when a byte frequency distribution is rank-ordered, it usually produces a Zipf-like distribution (exponentially decreasing frequency values). We rank order the distribution of a suspicious packet from most frequent to least and drop the frequency counts, resulting in a Z-String. A Z-String relies on the relative notion of frequency just by the ordering of the individual byte values, and since it is a string, we can apply the raw matching techniques described above to the Z-Strings themselves. Z-Strings are also often smaller than full packets (e.g., 8-bit byte-based packets would be referenced by a 256-byte Z-String), and as such the string comparison times are generally shorter than on the raw packets themselves. However, Z-Strings still have an $O(n \lg n)$ creation overhead in the size of the alphabet. (See section 1.3 for an example generated Z-String.)

Binary-modeled n-gram alert correlation

While frequency-modeled 1-gram alerts offer a measure of privacy, 1-gram modeling cannot represent a *sequence* of characters. For worms and other malicious binary payloads, we may want to capture such sequences, as they may serve as invariants across multiple suspect payloads that can be correlated. As discussed in [12], binary-based modeling produces surprisingly good results and leads to two different possible alert types.

N-gram signature. We can generate a list of n-grams that are found to be suspicious from an originating packet. Such a "signature" is position-independent while capturing specific malicious byte sequences. Given two n-gram signatures, we can simply compute the intersection of the two and threshold the cardinality of the intersected set to determine a similarity score. Such an intersection is linear time in the length of the signatures by using fast set-based data structures; depending on the n-gram size and packet content, this can vary significantly; while most packets are regular and have few n-grams, encrypted traffic, with a very flat byte distribution, can have as many n-grams as the size of the packet itself. In either case, an n-gram signature is a degenerate form of a raw packet; when distributing large n-grams, this is clearly not privacy-preserving, as even a 5-gram can contain a password. In these cases, we need a transformation on the n-gram itself.

Bloom filter n-gram signature. Instead of publishing an n-gram signature, we can instead insert the n-grams into a Bloom filter and publish it.[2] Since Bloom filters support both insert and verify, set intersections can be done between a (local) "raw" n-gram signature and a published BF n-gram signature, identifying the same n-grams as the previous technique *without* yielding other, potentially sensitive n-grams. This approach is also linear in time but leverages a BF's space efficiency. Optionally, *multiple* alerts can be published via a single Bloom filter, treating the BF as a *bag* of suspicious n-grams. This enables a multiplicative reduction in the amount of data transmitted and work needed to compute intersections.

Incidentally, correlating *two* BF n-gram signatures from different sites can be done via a bitwise AND "intersection"; this does not yield actual n-gram content, but may help find *commonality* between signatures, increasing confidence that the correct common code has been found when correlated against local data. BF intersection can also be used for model comparison, e.g., comparing two Anagram models to see if different sites exhibit similar traffic properties.

1.3 Results

Similarity Score As discussed in section 1.2, we compute a set of *similarity scores* for every correlation technique, $0 \leq score \leq 1$, with a higher score implying a more similar pair of alerts.

[2] This is *not* to be confused with Anagram's use of a BF model; here, individual alerts are placed into Bloom filters.

Raw packets and Z-Strings. For both of these alert types, our basket of string comparisons can be used. For SE, the score is binary: 0 or 1, where 1 means equality. For LCS and LCSeq, we use the percentage of the common LCS or LCSeq length out of the total length of candidate strings: $score = 2 * C/(L_1 + L_2)$, where C is the length of LCS/LCSeq and L_i is the length of string i. For ED, larger values imply dissimilarity; we normalize it as $score = 1 - D/(L_1 + L_2)$, where D is the computed edit distance and L_i the same as LCS/LCSeq.

Frequency distributions. As mentioned before, frequency distributions are compared using Manhattan distance: $M = \sum_{i=1}^{n} |x_i - y_i|$, $score = M/2$.

Raw and BF n-grams. Since we no longer have full packet content, we instead compute the percentage of common n-grams: $score = 2 * N_c/(N_1 + N_2)$, where N_c is the number of common n-grams and N_i the number of suspicious n-grams in alert i. If a Bloom filter is used, a count may be kept with it *or* approximated by N_b/N_h, i.e., the number of bits set divided by the number of hash functions used.

Testing with real traffic To compare the approaches, we randomly sampled HTTP packets from three sources: clean packets collected from *www* and *www1* (two heavily-trafficked Columbia CS webservers), and malicious packets collected from a sample of attacks (CodeRed, CodeRed II, WebDAV, Mirela, a phpBB forum attack, and an IIS buffer overflow (MS03-022) exploit). These packets were paired off in three sets: 10,000 "good-vs-good" pairs from 100 packets of *www* and *www1* traffic each, 1,540 "bad-vs-bad" pairs formed in the cross-product of the 56 packet malicious dataset, and 5,600 "good-vs-bad" pairs of *www1* and malicious packets. Similarity scores were generated for all of the resulting pairs with all techniques, except SE, which is too brittle to produce meaningful comparisons, and the n-gram analyses, which cannot be compared over an entire packet.

Figure 1 visualizes a small random subset (80 pairs) of the scores generated from the "good-vs-good" source. As figure 1 shows, the performance plots of the methods appear similar, although their centers and scale values differ as the scores are not normalized between the correlation methods. On raw payloads, LCSeq and ED bear very similar results, while comparisons on Z-Strings yield "flatter" results, as less information is compared.

As a more complete experiment, normalized scores were generated and compared for all of the pairs formed amongst the three datasets. To normalize the scores for a comparison, we first compute *similarity score vectors* V_A, V_B for the same data over two techniques A and B. The center of the two vectors are then aligned by shifting the median of V_A to match V_B. Finally, V_A's range is scaled proportionally so that its min and max values match V_B's. This normalization allows us to compute the Manhattan Distance of the two vectors, $distance = \sum_{i=1}^{n} |V_{A_i} - V_{B_i}|$; smaller values imply greater similarity between the two methods. Note that these scores are relative and dependent on the data used; the normalized results are only useful for comparing against a baseline, not as a source of absolute values or across datasets. These pairs were tested with each technique, and the resulting scores were normalized against and compared to the LCSeq score over raw packets. Table 1 shows the computed results.

Type	Raw-LCS	Raw-ED	MD	ZStr-LCS	ZStr-LCSeq	ZStr-ED
G-G	.0948	.0336	.0669	.2079	.0794	.0667
B-B	.0508	.0441	.0653	.0399	.0263	.0669
G-B	.0251	.0241	.0110	.0310	.0191	.0233

Table 1. Manhattan distance from Raw-LCSeq; lower is better.

Averaged over the three scores, Raw-ED is, unsurprisingly, closest to Raw-LCSeq. When privacy-preserving methods are considered, Manhattan distance performs the best overall, and particularly well for good-vs-bad comparison. All of the privacy-preserving methods are close when correlating pairs with attack traffic; we conjecture that significantly different byte distributions enable effective comparison even when some information is lost via privacy-preservation.

Cross-Domain Alert Correlation Next, we compare the techniques by examining their actual performance in identifying true alerts from false positives. Ideally, all false alerts are eliminated by a small similarity score (i.e. the site that produced the alert was the only site that saw this suspicious packet) while true alerts are identified with high similarity scores (i.e. the attack has been launched against more than one site).

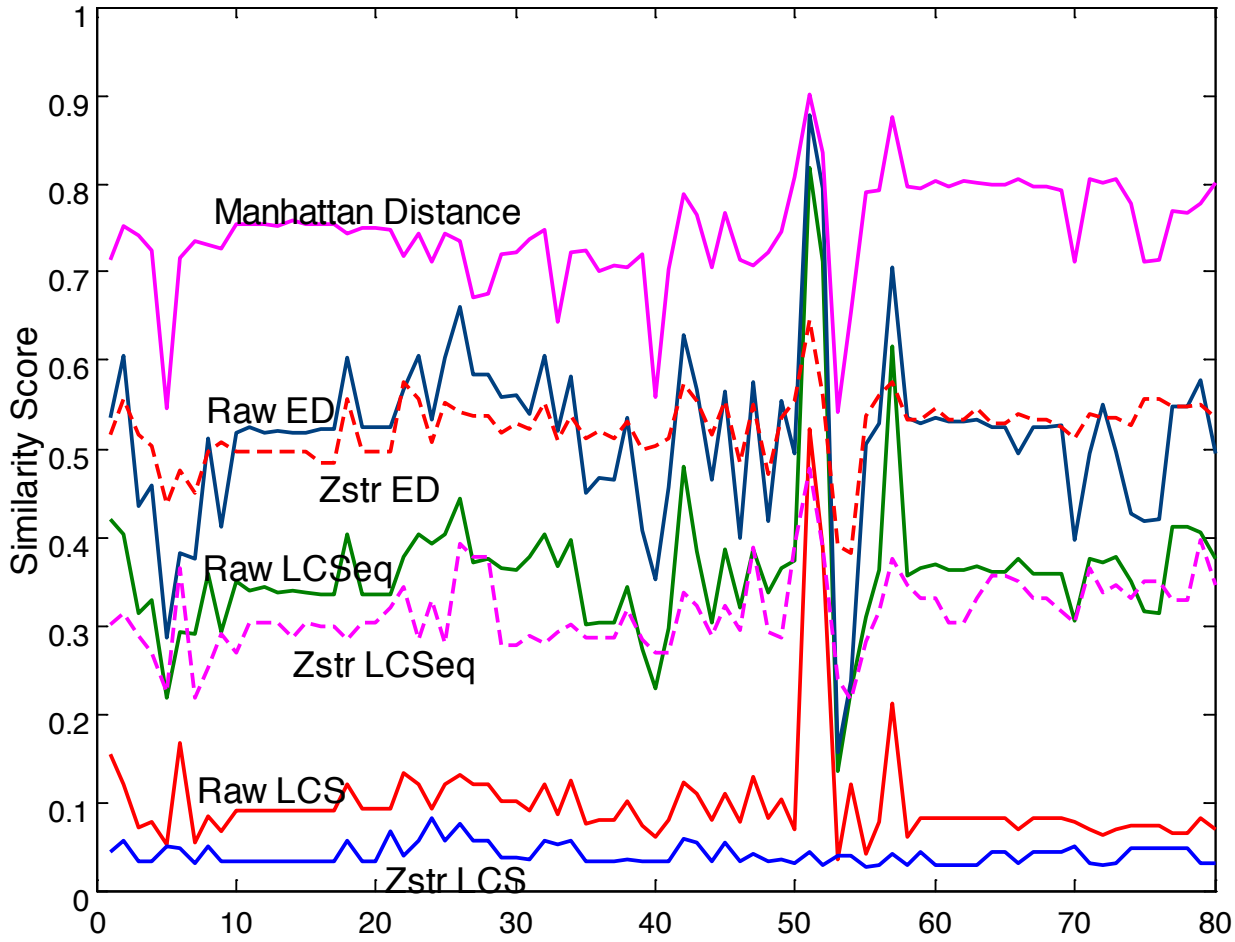

Fig. 1. Similarity score comparison of 80 random pairs of "good-vs-good" alerts.

In this experiment, we first randomly mix the aforementioned collection of attacks into two hours' traffic from *www* and *www1*, respectively. Multiple instances of attacks—4 for CodeRed and 3 for CodeRed II—are present to simulate a real-world worm attack. The attacks are also fragmented differently, as CodeRed does in the wild; for instance, CodeRed may fragment into a sequence of $(1448, 1448, 1143)$ length packets, $(4, 375, 1460, 1460, 740)$ length packets, etc. Multiple instances also enable testing correlation *between* different attack types (e.g., CodeRed vs. CodeRed II).

Next, the two mixed traffic sets are each run through PAYL and Anagram with previously-built models and with the alerting threshold lowered so that 100% of the attacks are detected, but with higher (and comparable) false positive rates. The resulting alert sets are correlated against each other using each of the techniques; the results are summarized in figure 2. For each method, the stacked bar represents correlation results for *false positives*. The shaded portion of the bar represents the 99.9% percentile similarity score range, while the white represents the worst-case (highest) score; in other words, while the worst-case FP score can be high, the vast majority of false positives score relatively low.

The asterisk-marked ("*") lines represent the range of similarity scores when instances of the same worm are correlated, and the open circle-marked ("o") lines represent scores across CodeRed and CodeRed II—a very simple measure of polymorphism. The other worms, which were inserted without fragmentation, all scored at or near 1, and so are not shown.[3]

We can draw several conclusions. First, correlation of identical (non-polymorphic) attacks works perfectly and accurately for all techniques. Most of the techniques can also correlate multiple instances of fragmented attacks; of

[3] We could have artificially fragmented these worms to simulate the CodeRed experiment, but we expect similar results.

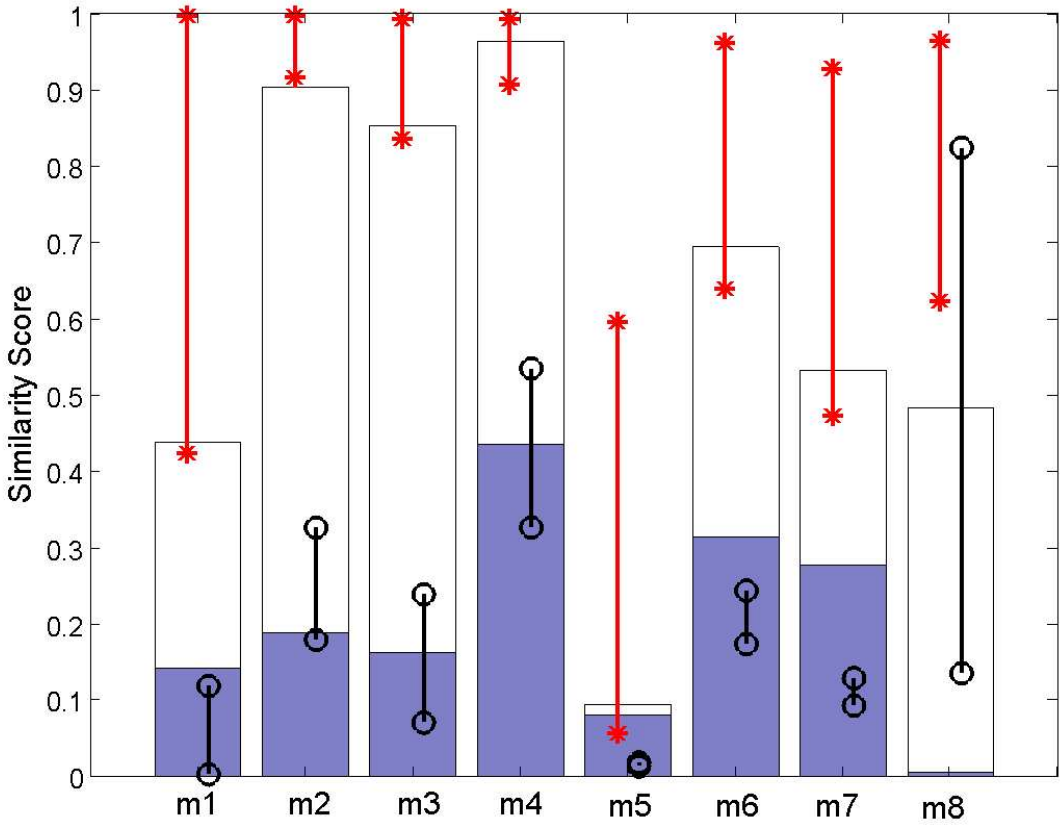

Fig. 2. Methods comparison. The correlation methods are, from 1 to 8, Raw-LCS; Raw-LCSeq; Raw-ED; Frequency-MD; Zstr-LCS; Zstr-LCSeq; Zstr-ED; N-grams with $n = 5$.

the privacy-preserving techniques, MD, LCSeq and ED on Z-Strings, and n-gram analysis[4] all perform well. (As intuition may suggest, ZStr-LCS is not particularly effective.) Polymorphic worm detection is far harder—even in the case of CR vs. CRII, only Raw-LCSeq and n-grams achieve promising results. N-gram analysis, in particular, stands out; it produces accurate results and is particularly effective at eliminating false positives, and the use of BFs enables privacy-preservation.

Signature Generation Correlating alerts across sites also enables the possibility of automatic signature generation and deployment, once true alerts are identified. (We can also potentially use the scores computed during similarity comparison as a "confidence" measure in mitigation strategies to determine whether to deploy a signature.)

Raw packet-based signatures. Given the ability to share raw alerts, we can exchange the LCS or LCSeq of highly similar packets. This has been the subject of much recent work, is not privacy-preserving, and we do not discuss it further here.

Byte frequency/Z-Strings. Given the first packet of a CodeRed II attack in figure 3 and its byte distribution displayed in figure 4, we can generate a Z-String by ordering the distribution by most frequent to least and dropping frequency information. Figure 5 shows the first 20 bytes of the generated Z-String for the distribution in figure 4, with nonprintable characters shown by their ASCII values. Both frequency distributions and Z-Strings can be used as signatures.

N-Grams. N-grams are an intriguing approach to signature generation; n-grams are position-independent, making them robust to reordering and fragmentation. Additionally, if position information is kept, such a collection *can* be transformed into a flat signature if desired. Figure 6 shows the results when a collection of 5-grams based on the

[4] We do not distinguish between published raw n-grams and published BF-based n-grams here, as they produce virtually identical results.

```
GET./default.ida?XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%u9090
%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%
u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u
00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u0
```

Fig. 3. Raw packet of CRII; only the first 301 bytes are shown for brevity.

CodeRed II example packet are "flattened". Nonprintable characters are represented by "."; "*" represents a wildcard for signature matching. Compared to the original, figure 6 successfully captures the malicious encoding and deemphasizes the padding "noise". Results with different n-gram sizes and another CRII packet are presented in an appendix in the extended version of this paper on our website [6].

2 Worminator/Whirlpool

2.1 Architecture

We adopt two mechanisms in order to cope with the difficulties of distributed correlation and the potential volume of data being correlated. First, the construction of Bloom filters by Worminator is employed to protect the confidentiality of the data being exchanged between domains. Second, efficient information exchange is accomplished with a distributed correlation scheduling algorithm. The scheduling algorithm dynamically calculates subsets of correlation peers that should communicate to exchange Bloom filters. Since information is also compacted by the Bloom filter, correlation between peers becomes extremely cost-effective in terms of bandwidth and processing power.

Requirements

1. The exchange of alert information must not leak potentially sensitive data.
2. Large alert rates hide stealthy activity; any reasonable solution must deal with or reduce the effects of these rates.
3. Centralized repositories are single points of failure and likely unable to correlate the burgeoning amount of alerts.
4. Exchanging alerts in a full mesh quadratically increases the complexity of the problem.
5. Any solution that partitions data among nodes risks information loss by disassociating evidence that should be considered in the same context.

We make several assumptions about the environment the system exists in and the alert information the system exchanges. Our assumptions and choices are intended to carefully balance the requirements of data privacy with the need to derive useful information and actionable intelligence from the alert exchange.

The environment and user base for a collaborative distributed intrusion detection system is an important consideration. We envision cohorts of 25 to 100 organizations exchanging information. Such cohorts can be organizations with similar interests, such as universities, financial institutions, military or government networks, energy companies, news organizations, *etc.*

The sheer volume of alert streams is a critical consideration in the design and evaluation of any distributed intrusion detection system. The size of current (and foreseeable) alert streams demands low–cost processing and correlation. Alert streams can threaten to dominate network bandwidth if they are unnecessarily replicated.

Perhaps the most important decision we make is to employ the use of "watchlists," or lists of IP addresses suspected of subversive behavior. The task of the distributed detection system is not to analyze the network or host events of other domains, but rather to correlate summaries of alerts to identify attackers. Therefore, watchlists encapsulate the appropriate information to exchange.

Preserving Privacy While IDS alerts themselves could be distributed, there are two substantial disadvantages to doing this: first, organizations may have privacy policies or concerns about sharing detailed IP data, some of which might uncover who they normally communicate with. Second, these alert files grow rapidly given substantial traffic.

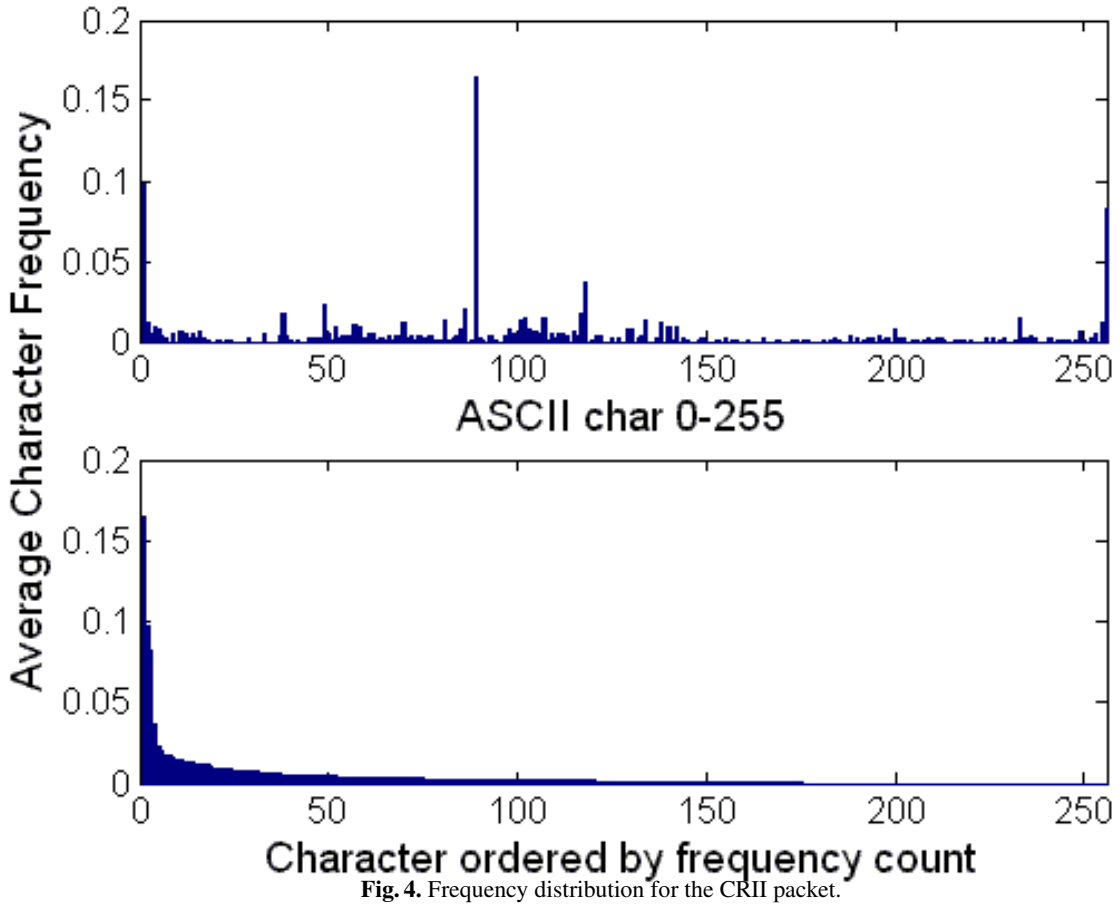

Fig. 4. Frequency distribution for the CRII packet.

```
88 0 255 117 48 85 116 37 232 100
100 106 69 133 137 80 254 1 56 51
```

Fig. 5. First 20 bytes of the Z-String computed from the CRII packet.

While parameters may be tweaked to reduce potential noise, a preferable solution would be to encode the relevant information in a compact yet useful manner.

We provide a compact format via the use of Bloom filters. A Bloom filter is a one-way data structure that supports two operations: insertion and verification, *e.g.,* while no data can be extracted after being inserted in the Bloom filter, it is probabilistically possible to see if specific data has been inserted if presented a second time to the Bloom filter. This is accomplished by creating a compact bit vector (typically between 2^{15} – 2^{20} entries). Entries are indexed by the hash of the original data, *i.e.,* a high-quality hash of original data (in this case, IPs and port information) is generated, broken up into parts, and these parts are used as indices into the bit vector. Each resolved index in the bit vector is set to 1. This process is typically repeated multiple times (for different parts of the hash and/or different hashes), thereby increasing resiliency to noise or data saturation. Verification is similar to insertion; instead of actually setting bits, the bit vector is examined to determine if the bits are already set. Therefore, the IDS parses its alert output and generates Bloom filters corresponding to (for example) IP/port endpoint data.

Since Bloom filters are compact one-way data structures, we get three benefits:

- *Compactness:* A Bloom filter smaller than 10k bits in size is still able to accurately verify tens of thousands of entries.
- *Resiliency,* even when the Bloom filter is decreased in size: When the Bloom filter is saturated, it starts giving false positives (*i.e.,* multiple data entries resolve to the same locations in the bit vector), but never gives false negatives. The false positives can be ameliorated by tuning or by correlation against multiple alert lists.

```
* /def*ult.ida?XXXX*XXXX%u9090%u6858%ucbd3%
u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%u
cbd3%u7801%u9090%u9090%u8190%u00c3%u0003%u8
b00%u531b%u53ff%u0078%u0000%u00=a HT*: 3379
```

Fig. 6. Generated 5-gram signature from the CRII packet; only the first 172 bytes are shown for brevity.

– *Security:* By utilizing a one-way data structure, organizations can correlate watchlists without releasing actual IP data, satisfying privacy needs while being able to participate. If further security from outside observers is needed, the dissemination protocol can be encapsulated in a secure tunnel, like SSL, thereby only granting Bloom filter access to the set of participants in the alert list correlation.

Distributed Correlation A distributed correlation function must overcome the problems of a centralized model while balancing the information loss inherent in partitioning alert data among different nodes. The most straightforward way to accomplish distribution (forwarding all data from every node to every other node) involves a quadratic increase in the amount of data exchanged.

More sophisticated approaches are based on two different theories. The first approach, which mirrors traditional DHT-based P2P networks, creates an explicit mapping from alert data (specifically, source and target IP addresses) to particular correlation nodes. The reasoning behind this approach is that source and target IP addresses are the two most important features (besides target port) of alert data. With data about various machines collected in one node, the majority of correlation can be accomplished at that node without communicating with other nodes (except to distribute results of global interest).

The shortcoming of this approach is that nodes become special cases of the centralized model: they are single points of failure for information pertaining to the IP address range being hashed. In addition, participants in the system may be uncomfortable with storing their raw alert information at a single node. This approach invests too much trust in each node. While a self–healing approach like Chord [10] can ameliorate the loss of a node, previous information stored at that node is at best lost temporarily (for example, in the case of a denial of service,) or corrupted (in the case of the node being compromised). Some of these shortcomings can be mitigated by replicating the data to some number of other nodes; however, it is not clear what the appropriate balance is between fault–tolerance through replication and utilization of network bandwidth and storage space. If data is replicated to every other node, we see an unacceptable quadratic increase in the cost of the system. Furthermore, while DHT-based overlays provide a fast *lookup()* operation, the performance of such networks under churn (rapid series of *join()* and *leave()* operations) is questionable [3].

The second theory attempts to address the limitations of the first approach by introducing a dynamic mapping between nodes and content. This *dynamic overlay network* (as opposed to the largely static mappings of traditional DHT-based overlay networks) implicitly incorporates the notion of churn and does not need to spend time rebuilding neighbor (finger) tables. We observe that a theoretical optimal schedule exists for communicating information. If an oracle existed in the network that answered with the appropriate subset of nodes that should talk given a particular alert, links could be established between these nodes without talking to nodes with irrelevant data (*e.g.*, without a *lookup()* operation).

In this model, we assume that there is a set of nodes S of size N. We assume that there is some reliable mechanism for any subset of nodes to communicate with each other. There is some discrete unit of knowledge K, that if known would provide evidence of a distributed scan. During a distributed scan, some subset of S is scanned and now contains a piece of knowledge K_i.

Normally, this knowledge would be discarded as insignificant. However, the optimal schedule allows this set of nodes to perfectly guess which of its neighbors also contains a piece of K. Note that each node could also find this information out via the $O(N^2)$ full mesh method of asking each other node in the network. However, we assert that this method is too costly in terms of trust, network bandwidth, and disk storage.

The key idea in this optimal schedule is that the correct subsets of nodes are always communicating. In order to mimic this behavior, the (approximately) correct nodes must talk to each other at (approximately) the right time. One way of accomplishing such a schedule is to pick relationships at random. Another way is to employ a publish-subscribe scheme.

Whirlpool: Network Scheduling There is a clear need for efficient alert correlation in large–scale distributed networks. To address this need, we introduce the notion of *network scheduling:* the controllable formation and dissolution of relationships between nodes and groups of nodes in a network. These relationships can be envisioned as a dynamic overlay. Our network scheduling mechanism is a procedure for coordinating the exchange of information between the members of a correlation group. The mechanism is controlled by a dynamic and parameterizable correlation schedule.

Our approach is predicated on the previously described theoretical model of the optimal correlation schedule, the shortcomings of a fully interconnected mesh of correlation nodes, and the limitations of the ideal centralized approach to large–scale correlation. The main difficulty is that nodes would most likely discard data that in truth belong to a distributed alert. We must develop a mechanism whereby a node can quickly conference with other peers and determine whether or not a local alert is noise or signifies part of a distributed alert.

The basic architecture of network scheduling is a set of dynamic federations. Nodes that join and leave these federations at various rates. The variance in rates is intended to allow federation groups to retain some stability while expediting the import of new information into the group. Furthermore, this mechanism can be augmented with a distributed learning algorithm that assists in promoting alerts discovered by the distributed correlation.

2.2 Implementation

Worminator The Worminator platform supports compact watchlist correlation via the replication and use of Bloom filters [1] and uses the Antura network intrusion detection system [8] to generate alerts. The Antura NIDS has been demonstrated to be an order-of-magnitude better at detecting long-term stealthy scans than competing products.

Our initial proof-of-concept version of Worminator ran at specific intervals on the computer running the Antura sensor, parsed its alert output, and generated Bloom filters corresponding to IP/port endpoint data. These alerts were then transmitted to a centralized node using HTTP, and correlation was manually initiated after-the-fact by downloading these Bloom filters from the centralized node. This proof-of-concept prototype demonstrated the feasibility of the idea: in preliminary tests that correlated results from installations of Worminator at two academic sites, three common sources of stealthy surveillance were detected: one each in Beijing, the Phillippines, and a small western US community college. These sources are most probably interested in the test sites to discover weakly protected (but usually more powerful and better connected) university or research machines.

We have since evolved this version to better handle communication latency and privacy requirements. The current version of Worminator is completely pluggable, and supports different sensor and alert types, correlators, and communication frameworks. Just as importantly, it supports *continuous validation*; alerts from the sensor are exchanged immediately, and correlation runs real-time to glean data as soon as possible to help prepare defenses against pending attacks or fast-moving worms.

The goal is to enable sites to maintain a secure *watchlist* of alerts seen locally and from other sites, and to generate a *warnlist* of significant threats if they have been correlated as having been seen at multiple sites. This warnlist can then be reported to network administrators or could be directly mapped to firewall rules to prevent impending attacks. Depending on privacy policies, these local warnlists may also be explicitly replicated to other sites to enable a fast global-scale response.

Worminator consists of approximately 9,500 lines of Java code, and leverages a number of J2EE (Java 2 Enterprise Edition) providers, including the PostgreSQL JDBC provider (for querying databases), the Tomcat JSP/Servlet container (for the user interface), and the JBossMQ JMS provider (for event transport).

2.3 Results

Worminator collaborative site experiment The new version of Worminator was deployed at four different sites in the Northeast: two at Columbia (one on the perimeter of the Computer Science network and the other on the perimeter of a campus dorm), one at a company in midtown Manhattan, and the last a research institution in Washington, D.C. The Antura sensor was used as the source of alerts at each site, and Java Message Service, a publish-subscribe communication infrastructure, was leveraged to support event distribution amongst the sites. This configuration was run for approximately 96 hours. Information on source IPs and scan times were exchanged.

During the time, approximately 550,000 alerts were generated and exchanged. By far the most of these alerts (roughly 75%) were at the Computer Science network. In addition to traffic passing through the perimeter of the CS network, the CS sensor sees internal traffic between CS machines. (Ordinarily, the dorm network might be more

Source	# alerts	Alerts/hr	# src IPs
CS	420425	4379	5226
Dorm	9838	204	290
Midtown	11394	118	1831
Wash DC	116560	1214	22568

Fig. 7. Statistics on the *watchlist*: Sites, number of alerts exchanged, and number of source IPs detected.

saturated with malicious traffic, but this data was taken during Spring Break, when many undergraduates leave campus. The dorm sensor was also deployed later than the other sites, and had run for approximately two days.)

As Figure 7 implies, IPs generated more than one alert in some instances; nevertheless, the number of IPs is very large for each site, making individual alerting difficult. A total of 29,731 unique IPs were seen. Next, we ran queries to determine which sources were seen at two, three, or four sites.

# of sites	# common src IPs	% reduction
Two	170	99.5%
Three	18	99.93%
Four	1	99.996%

Fig. 8. Statistics on the *warnlist*: the number of source IPs detected at two, three, and at four collaborating sites.

The reduction by examining the set of common sources is remarkable – only 18 of the 29,731 original source IPs were observed at three sites. While this does not necessarily discount the other 29,713 IPs, the likelihood that legitimate traffic exists between three of these four unrelated sites is extremely low, and simple port analysis can help confirm this hypothesis. Note that, due to reduced activity at the dorm site, conclusions about the number of common sources at all four sites is preliminary at best, but the one site that matched originated from China. Further analysis from the CS sensor revealed that the machine was probing destination ports 1026 and 1027 – used by the Windows Messenger service [4], strongly suggesting this IP was doing wholesale "pop-up" Internet spamming. While its activity *may* have been benign, such a source is clearly indicative of undesirable traffic, and leveraging the warnlist makes it easy to block such sources, be it undesirable or outright malicious.

We also performed some preliminary analysis on the two-site data to determine if a significant geographic distribution existed (e.g., if more scans originated from any particular site). Figure 9 illustrates the results of the top eight countries (which comprise 87% of the total scans – every other country contributed two or less IPs to the overall total).

As the chart shows, China dominates, with nearly 49% of the top eight countries and 41% overall amongst all countries. This suggests that, despite the "great firewall of China", outbound scans and probes are unaffected and continue to propagate to a broad cross-section of the Internet.

One interesting footnote: while performing analysis on the two-site data, two alerts were generated for the IP `128.9.168.45`. Reverse DNS revealed the URL to be `http://ptr.isi.edu`, which turns out to be an Internet mapping server performing "low-volume" scans. Indeed, this source comprised only .00007% of the total alert exchange, yet we were able to focus on the source with a minimum of effort.

This analysis only scratches the tip of the iceberg; it becomes clear, however, that useful data can easily be gleaned with just a few sites exchanging alerts. We plan to increase the number of participating sites, which will greatly increase the depth and breadth of the types of sources of surveillance and we expect it to further validate our approach.

Alert Rates Intrusion detection systems face the very real threat of information loss from the sheer rate of available information. Schaelicke *et al.* [9] are decidedly pessimistic about the ability of relatively powerful commodity hardware and network links to absorb peak alert loads, noting that an IDS is effectively neutralized by the loss of alert data resulting from a database unable to keep up with incoming network data. This problem is compounded if multiple NIDS sensors report to the same database system. DShield reports about 10 million alert records added daily. Figure 10 shows the increase in contributed data per month between January 2002 and May 2003.

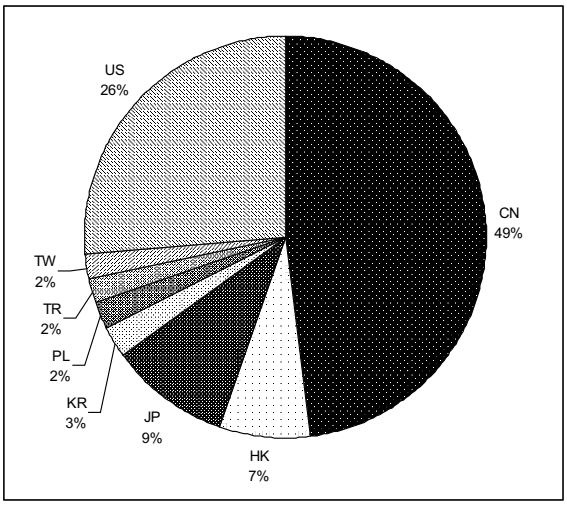

Fig. 9. Geographic distribution of attacks by country for 2-site-detected scans.

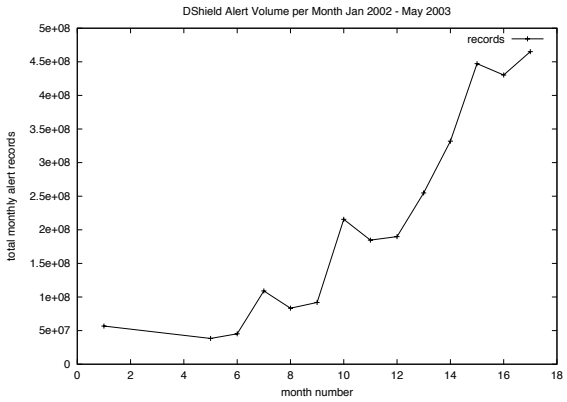

Fig. 10. DShield monthly alert record contributions. The graph is not cumulative, but rather shows the rapid increase in contributed alert information per month as DShield grew in popularity.

Event reduction and aggregation is a critical part of our system. Since we construct watchlists from very little information (an IP address and a port), we are interested in ways of combining different alert information. Reduction can be accomplished by filtering events either at the source sensor or prior to correlation processing. As a trivial example of the latter, consider a series of 100 basic IDS alerts with the same source and destination IP information and alert type information (*e.g., "Host X probed host Y on port Z"*). These alerts can be reduced to a single alert and a frequency. If a significant portion of alert streams are amenable to this type of reduction, we can either perform more expensive processing on the resulting stream, or produce actionable intelligence more rapidly. Inexpensive reduction strategies (like logically grouping attacking IP source addresses in the same /24 subnet) can result in substantial compression, as is aggregation of multiple scan alerts (one per port) by a single source into one overall alert announcing a scan. For an example of the successful application of these reduction strategies,

Network Schedule Evaluation To evaluate the effectiveness of the Whirlpool network scheduling, we compare it against a full mesh distribution scheme and a random selection distribution scheme. To that end, we introduce a *Bandwidth Effective Utilization Metric* (BEUM). The BEUM is defined as:

$$BEUM = \frac{1}{t * B}$$

where t is the average number of time units it takes the distribution scheme to detect an attack and B is the amount of bandwidth used by the distribution mechanism during that period. B is defined in terms of the total number of nodes, N, communicating via the distribution mechanism. Thus, for a full mesh scheme we have:

$$B = N * (N - 1)$$

and the time to discover an attack is $t = 1$. The BEUM for a full mesh distribution strategy is therefore $\frac{1}{N*(N-1)}$. For a system of 100 nodes, the BEUM for a full mesh is $\frac{1}{9900}$

The BEUM for a particular schedule where groups are kept at roughly $\sqrt{N}$ is different and based on the calculation of the bandwidth consumed, B:

$$BEUM = \frac{1}{t * B}$$

$$B = N * \sqrt{N}$$

In general, if $t \leq \sqrt{N}$, this particular schedule wins. Specifically, for a system of 100 nodes, the BEUM is $\frac{1}{1000t}$. If $t \leq 9$, this schedule is a better choice than a full mesh. Many other schedules are possible to balance the tradeoffs between bandwidth, coverage, and latency and we are exploring methods for identifying optimal schedules given a set of constraints.

We simulated a randomized scheduling strategy for a system of 100 nodes (performed over 1000 trials). Our simulations indicate that on average, it takes 6 time units before an attack is detected using a repeated random schedule. This time unit requirement satisfies the requirement for $t \leq 10$ we derived for the BEUM. Figure 11 shows that even though some pathological outliers exist, the vast majority of attacks are detected in a relatively short time.

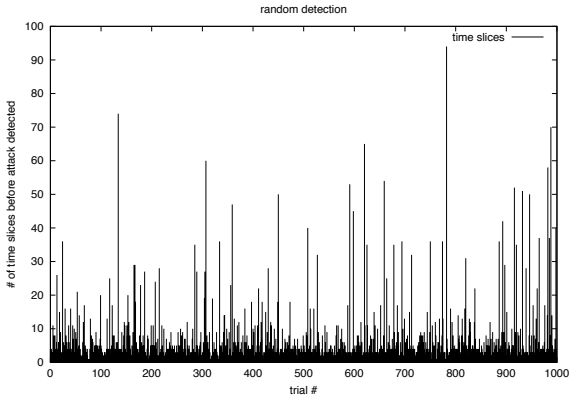

Fig. 11. Number of time slices until random distribution detects an attack. The average number for this particular data plot is 6 time slices.

3 Longitudinal Study of Scan Behavior

This study of stealthy scan behavior is designed to demonstrate the proposed Worminator hypothesis, that collaborative intrusion detection not only enables detection of *worm spread* but also scanning behavior as precursors to an attack. There are three key *longitudes* for analysis:

1. **Over time**: as it is difficult to determine ahead of time when a widespread worm attack will occur, the goal here is not necessarily to correlate certain scan behavior with particular attack behavior[5], but to identify long-term

[5] Correlating scan and attack behavior is difficult, unless known attacks occur during the scan period. While we collected much interesting information, major Internet-scale attacks did not occur during the Worminator experiment, and so conclusions are not predicated on such a correlation.

scanners who try to "fly under the radar" by throttling their scanning rates at any individual site to as little as a few scans per day per IP. One can define a measure of *stealthiness* by looking at scan time windows, both on a local (single-site) and on a global (many-site) basis.

2. **Over geographical and network space**: a clever attacker is unlikely to focus all their scanning efforts from a single source; instead, the current trend is to spread scan efforts over a broad range of sources, and to leverage those sources as a proxy to mask scanning behavior. Botnets ([5], [7]) are also becoming an increasingly common tool for scans. By leveraging collaboration, the goal is to observe a wider destination space to ease detection of broader networks of coordinated scanners.
3. **By target**: one key form of anonymity described in this thesis is that of *anonymous but categorizable*. This allows for the exploration of *targeted scans*, e.g., sources that scan particular categories of networks but skip others. Here, we compute aggregate statistics on the popularity of commercial vs. academic targets, etc. As of the writing of this thesis, only two commercial sites have been deployed, so the results of this experiment are limited.

3.1 Scan Lengths and Stealthiness

As the Worminator system was design to observe long-term scanning behavior, the first question of relevance is the actual scan behavior of sources, especially those who are observed at multiple sites. Table 2 shows aggregate scan length results for sources (site/IPs, not site/IP/destination tuples) appearing at exactly 1 through 5 sites, in days, e.g. source IPs seen at four sites were observed, on average, for a period of about 30 days.

# Sites	# (Site,IPs)	Max	Avg	StDev
1	307050	373.57	7.14	33.12
2	22250	372.60	10.86	36.98
3	10074	373.64	17.20	47.01
4	3228	373.65	29.86	60.09
5	245	373.49	70.77	102.15

Table 2. Maximum and average scan lengths for 1–5 sites, by source IP/site, in days.

The conclusion is clear: sources which are observed at multiple sites tend to scan for longer periods. The most likely explanation for this behavior at the small scale is the elimination of false positives; source IPs that are seen at two or three sites often eliminate the local false alerts that IDSes typically observe. On the other hand, the dramatic increase in average for 5 sites is interesting. Figure 12 shows a time plot of the scan lengths for sources that scanned 5 sites.

This suggests that, indeed, many 5-site-scanners were long-term, and that the high standard deviation is primarily due to the limited length of the experiment (and the fact that not all sites were up for extended periods of time). Unfortunately, conclusive results cannot be drawn from the small sample set, but still, the noticeably higher average scan time suggests that many of these sources are long-term broad scanners—and that corroboration helps to identify them.

These results do not take *volume* into account, however. In particular, if a scanner happens to be a machine that aggressively scans all of its targets, that's more easily detectable without corroboration. Of greater concern are scanning sources that only generate a *few* alerts at each site over a long period of time. These scanners essentially fly under the radar by hiding behind all the noise generated at individual sites. By corroborating and looking for the slowest scanners over long periods of time, we can find, without difficulty, entities who are looking to do significant machine mapping.

To do this, we define a "stealthiness" metric St for any arbitrary source s_i, total scanning time t_i and number of alerts $|a|_i$:

$$St(s_i) = \frac{|a|_i}{t_i}.$$

Low stealthiness levels amongst scanners at multiple sites is of particular interest; tables 3 and 4 show the top-10 stealthiest scanners detected across 4 and 5 sites, respectively.

As can be observed, there are scanners that issue only a few scans per site over the course of a year. Even more interesting are the italicized entries—these are scanners from the same subnet! A quick lookup on that /24 yields the results in table 5. ev1 is a major ISP in the United States, and this may have been the IP space of one "customer" (be

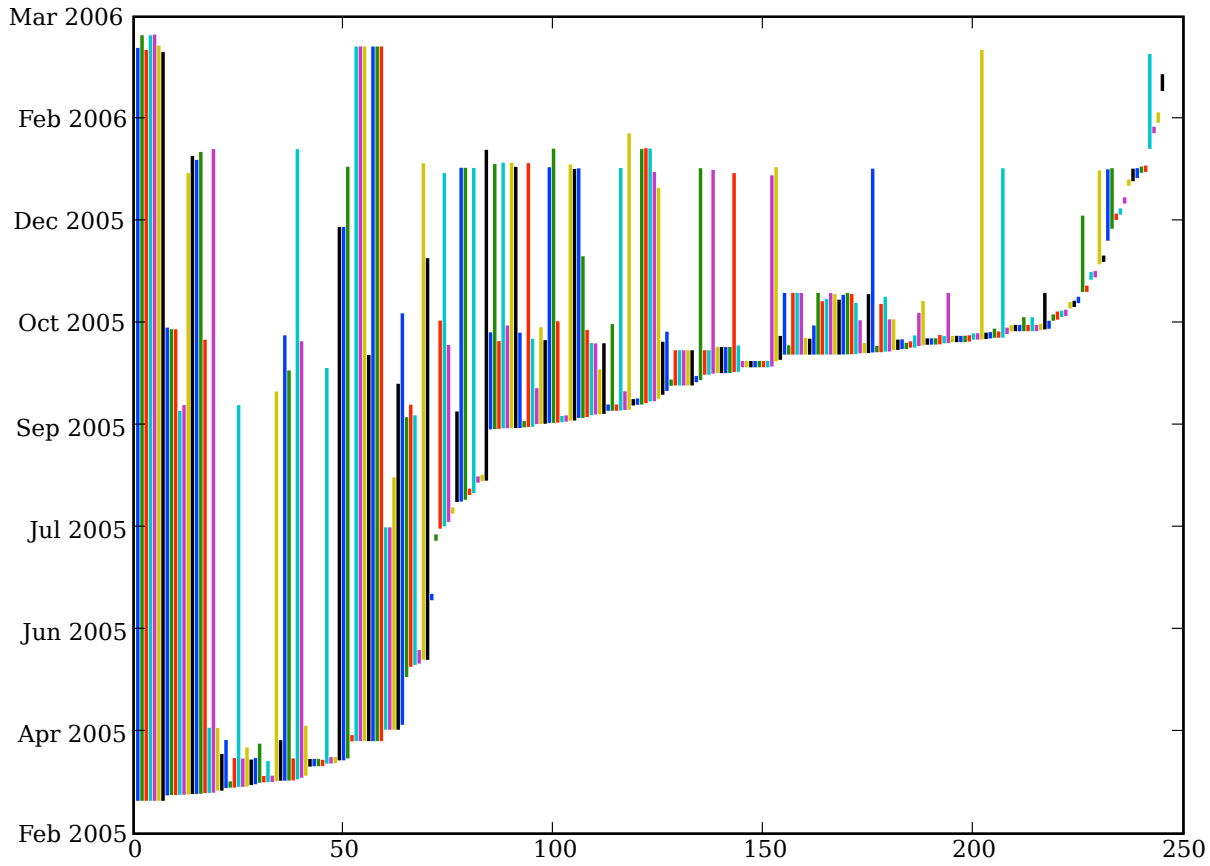

Fig. 12. Scan length distribution, 5-site scanners.

it a legitimate customer whose machines were subverted against their knowledge, or an illegitimate customer using the machines as a scanning source). The likelihood that these hosts were legitimately present at 5 disparate sites is extremely unlikely, especially since several of the sites have absolutely no relationship with each other (excepting this study; however, no Columbia IPs are listed above).

Further discussion about subnet analysis can be found later in this subsection.

3.2 Breadth and (Loud) Volume

As a counterpart to the previous subsection, Worminator should also ideally be able to identify the *noisy* sources—to enable, for example, evidence of an active attack. There are various ways to establish a noisy source, including: the aforementioned stealthiness metric can be used to determine the *least* stealthy source; the number of alerts generated by the IDS may also serve as an indicator, regardless of scan length; and the number of sites a source appears at. Figures 6–14 show the *noisiest* sources at 4 and 5 sites using the stealthiness and alert count metric, respectively.

The results shown here, especially the ones by stealthiness, are remarkable; for example, the top noisiest source issued 331 alerts scattered amongst 5 sites over the space of two days; a quick port analysis yields that all of these were to port 22 (ssh), suggesting a brute-force password attack against ssh servers. Moreover, a number of IPs in 61.152.* appear in the top 10 by *both* noise metrics. Of particular note was 61.152.158.109, which generated nearly 2,000 alerts over the stretch of three months at all five sources. A quick port analysis yields that these scan alerts were distributed across ports 1026–1030, which is indicative of a Windows Messenger spammer. (As mentioned before, active large-scale worm attacks were not observed during this period, but one can construe a UDP spammer as an attacker, as scanning behavior will likely be similar.)

Source IP	Scan Length (days)	# Alerts	St
61.185.246.34	257.73	7	3.144e-07
207.218.223.98	302.96	9	3.438e-07
61.129.45.54	302.12	10	3.831e-07
207.218.223.91	270.71	9	3.848e-07
207.218.223.89	271.16	11	4.695e-07
207.218.223.93	301.50	13	4.990e-07
66.150.8.18	199.92	10	5.789e-07
62.189.244.254	287.28	17	6.849e-07
61.172.250.90	234.36	14	6.914e-07
206.253.195.10	293.14	19	7.502e-07

Table 3. Top 10 stealthy scanners detected at 4 sites.

Source IP	Scan Length (days)	# Alerts	St
207.218.223.92	300.14	12	4.628e-07
207.218.223.103	302.52	17	6.504e-07
69.7.175.21	293.50	41	1.617e-06
69.25.27.10	225.52	33	1.694e-06
161.170.254.232	299.29	51	1.972e-06
219.148.119.199	227.03	45	2.294e-06
66.151.55.10	303.12	62	2.367e-06
62.73.174.150	338.39	90	3.078e-06
64.41.241.171	338.39	90	3.078e-06
64.56.168.66	338.39	96	3.283e-06

Table 4. Top 10 stealthy scanners detected at 5 sites.

Given such metrics, a simple thresholding may enable automatic response with high confidence, which is ultimately what is desired during an actual attack. Therefore, in addition to determining stealthy *scanners*, we can also identify active *attackers*, to enable a comprehensive two-pronged approach.

3.3 Geographic Analysis

Given multiple-site corroboration, we can do some analysis to see if there is any correlation between multiple-site scanners and geographic tendencies, by both the number of scanning sources and the number of alerts generated by IDS sensors. A combination of DNS and WHOIS data was used to determine the geographic distribution of IP addresses. Figures 15–19 show the results of this analysis. The country codes shown are the ISO codes used by WHOIS. Countries with less than 1% of alerts or IPs are not shown, and are instead lumped into "Other".

Source IP	Scan length	#alerts
12.130.50.213	373.49	5978
12.130.50.214	373.47	5589
61.152.91.69	56.19	3498
61.152.239.68	321.26	3311
219.138.199.170	47.73	2769
218.30.114.214	259.45	2436
70.86.131.171	48.94	1989
67.182.20.245	315.31	1869
68.114.241.56	307.77	1865
66.38.27.13	369.60	1853

Fig. 13. table
Top 10 noisy scanners by # alerts, 4 sites.

Source IP	Scan length	#alerts
212.176.49.56	302.58	3697
61.152.158.109	93.84	1920
69.133.97.207	364.95	1654
24.164.180.228	142.62	1300
199.97.98.40	368.12	1024
128.9.160.82	373.18	1017
128.9.160.251	373.49	1016
82.77.62.33	366.00	1012
128.9.160.83	373.18	1011
81.74.106.18	42.45	504

Fig. 14. table
Top 10 noisy scanners by # alerts, 5 sites.

Source IP	#sites	#alerts	Scan len	Hostname
207.218.223.92	5	12	300.14	ivhou-207-218-223-92.ev1servers.net
207.218.223.103	5	17	302.52	ivhou-207-218-223-103.ev1servers.net
207.218.223.89	4	11	271.16	ivhou-207-218-223-89.ev1servers.net
207.218.223.91	4	9	270.71	ivhou-207-218-223-91.ev1servers.net
207.218.223.93	4	13	301.50	ivhou-207-218-223-93.ev1servers.net
207.218.223.98	4	9	302.96	ivhou-207-218-223-98.ev1servers.net
207.218.223.94	3	10	300.44	ivhou-207-218-223-94.ev1servers.net
207.218.223.95	3	8	301.51	ivhou-207-218-223-95.ev1servers.net
207.218.223.97	3	8	63.06	ivhou-207-218-223-97.ev1servers.net
207.218.223.99	3	10	271.10	ivhou-207-218-223-99.ev1servers.net
207.218.223.102	3	10	297.12	ivhou-207-218-223-102.ev1servers.net
207.218.223.90	2	9	20.04	ivhou-207-218-223-90.ev1servers.net
207.218.223.101	2	5	270.55	ivhou-207-218-223-101.ev1servers.net
207.218.223.100	1	1	3.99	ivhou-207-218-223-100.ev1servers.net
207.218.223.132	1	4	2.12	ns1.rackshack.net
207.218.223.162	1	6	1.05	ns2.rackshack.net

Table 5. Subnet search results for 207.218.223.0/24.

Source IP	Scan Length	# Alerts	*St*
61.143.210.244	0.07	137	2.224e-02
64.246.36.36	0.03	58	2.118e-02
58.215.64.204	0.03	45	2.022e-02
211.100.17.210	0.03	42	1.829e-02
202.103.178.214	0.02	33	1.817e-02
220.166.63.45	0.22	318	1.662e-02
211.76.177.154	0.13	174	1.567e-02
58.215.65.43	0.05	64	1.512e-02
222.191.251.92	0.09	111	1.488e-02
219.149.86.90	0.03	33	1.442e-02

Table 6. Top 10 noisy scanners by stealthiness, 4 sites.

The trend from 1-site to 4-site is clear: as more sites' alerts are corroborated, the geographic distribution takes an increasingly international bent; most notable is the shift from the US being the primary source of alerts to China. Part of this is due to the fact that corroboration eliminates most of the false positives observed at local networks; for example, most false positives in CUCS would be attributable to machines on the same LAN. This is already visible in the second chart in figure 15, where the US actually has significantly more alerts than actual IP space. By the time 4-site scanners are counted, China has 45% of total alerts and 51% of IP addresses. 5-site scanners are somewhat of an anomaly to this trend. However, the 5-site dataset is small, and it is difficult to draw concrete conclusions from it. Further study is required to see how this trend continues when more sites are included (although it should be noted that Russia seems to have a significant presence amongst 5-site scanners). What remains clear is that the largest source of scan behavior emanates, by far, from two primary countries. This also suggests that, despite the "great firewall of China", outbound scans and probes are unaffected and continue to propagate to a broad cross-section of the Internet.

3.4 Scanning Subnets

As mentioned earlier, the presence of multiple scanners within the same subnet may be indicative of a coordinated scan or attack, which may be of greater interest especially if multiple sites have seen the same behavior. To evaluate this, a scan was made through the IP addresses collected through the IDS to see if any interesting outlier class C subnets (i.e., /24s) were found. To make this more accurate, the "number of sites" criterion was modified to act as a lower bound, since not all addresses may have been detected at all sites. Table 8 shows the aggregate statistics for subnet scanners.

On *average*, very few IPs ($\approx 1-2$) are found to be scanners within any given subnet. This would imply that a subnet with 115 scanners detected on at least *two* sites is a significant anomaly—60 standard deviations above the mean, to be precise! To get a better feel of the outlier distribution of scanning subnets, figure 20 shows a logarithmic

Source IP	Scan Length	# Alerts	*St*
149.205.192.85	1.85	331	2.069e-03
61.141.32.80	11.08	484	5.055e-04
61.152.158.109	93.84	1920	2.368e-04
212.176.49.56	302.58	3697	1.414e-04
81.74.106.18	42.45	504	1.374e-04
162.40.95.86	48.64	456	1.085e-04
24.164.180.228	142.62	1300	1.055e-04
69.40.165.231	33.58	282	9.720e-05
207.67.25.104	61.45	290	5.462e-05
69.133.97.207	364.95	1654	5.246e-05

Table 7. Top 10 noisy scanners by stealthiness, 5 sites.

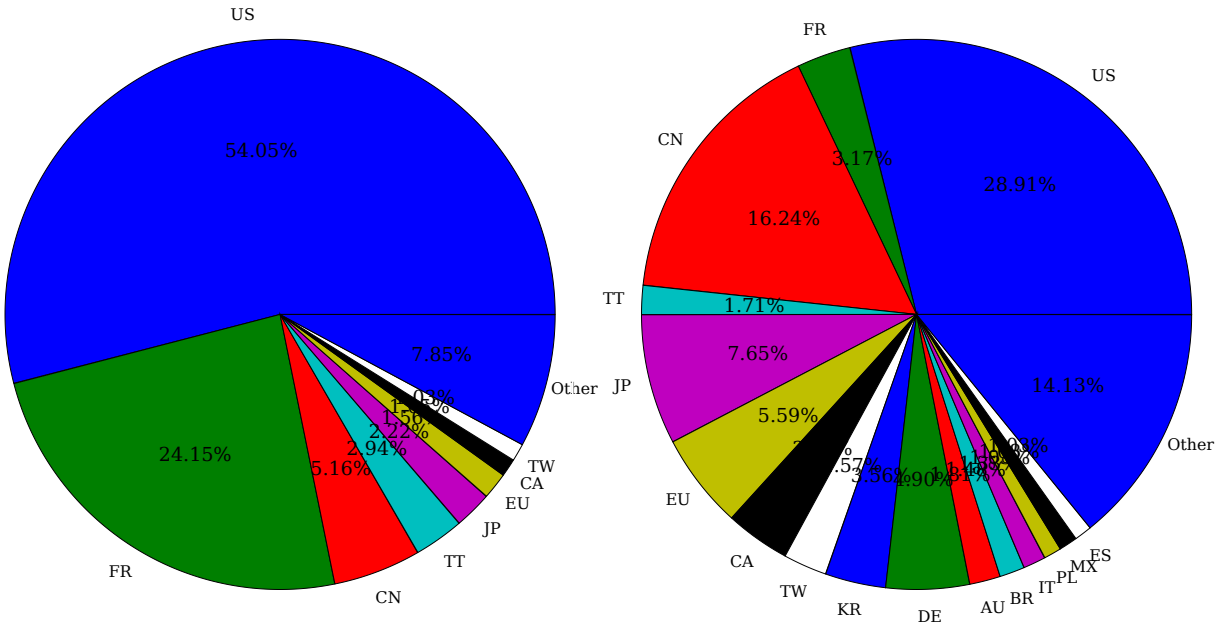

Fig. 15. Geographic distribution of 1-site scanners, by # of alerts and # of IPs.

graph of the largest subnets varying with the minimum number of sites the scanners are detected upon. While all of the variations show a "long tail", as more sites are involved the "head" of the tail becomes a larger outlier. While the possibility remains that these IPs were independent and coincidentally happened to be many active blocks in the same class C, the statistics make this extremely unlikely.

It is worth mentioning that newer scanning *botnets* are not necessarily restricted to single subnets; indeed, many of the newer scanning approaches use a much broader range of machines, such as compromised computers distributed across the Internet.

3.5 Target Analysis

One last significant form of analysis is looking for alerts that correspond to certain target *longitudes*, i.e., scanners that may target commercial institutions but not academic institutions, or vice-versa. These may be indicative of scan sources that are more than just purely automated—they may be actively scanning some entities and not others to build more specific hitlists.

Given our collected data, a total of 2,095 sources matched these criteria; 311 sources targeted *all three* academic institutions but neither of the two commercial ones, while 1,784 sources targeted *both* commercial institutions but none of the three academic sites. Tables 9–12 show the top 10 for each, measured by number of alerts and stealthiness, respectively, along with the top ports for the sources in question. A number of these correspond to well-known services;

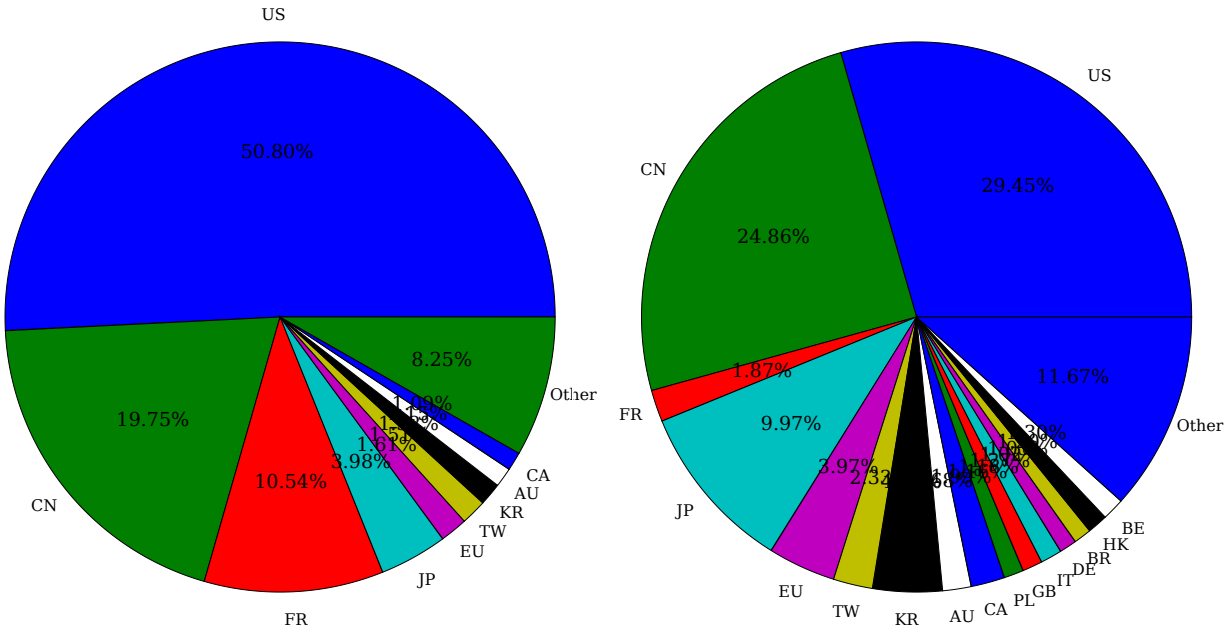

Fig. 16. Geographic distribution of 2-site scanners, by # of alerts and # of IPs.

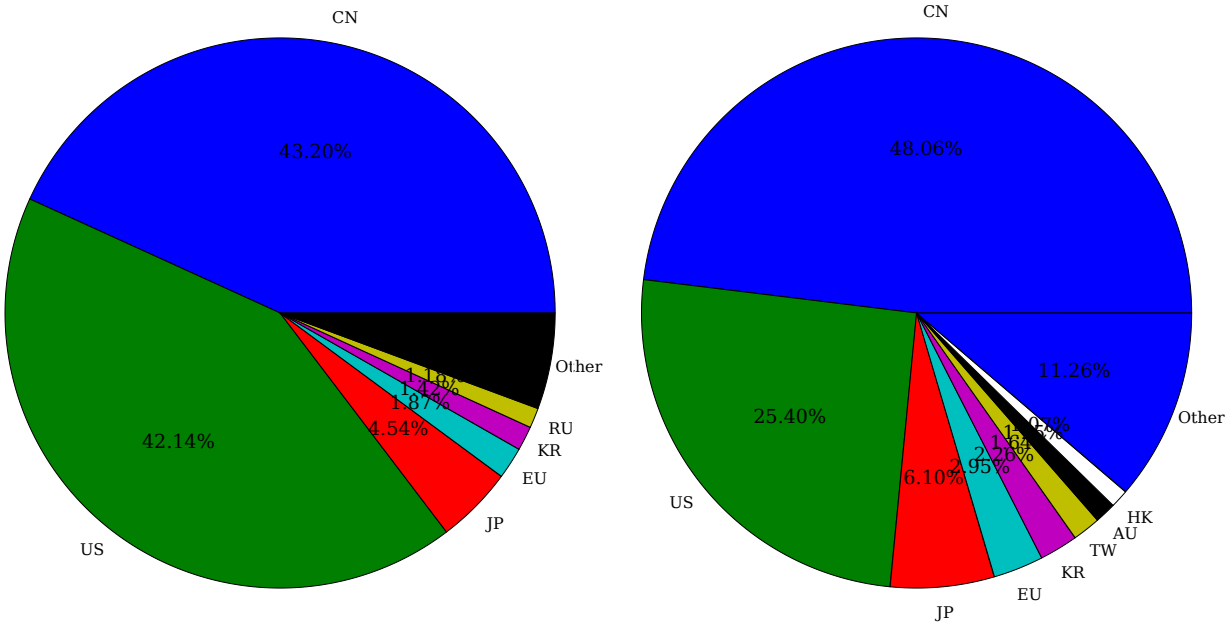

Fig. 17. Geographic distribution of 3-site scanners, by # of alerts and # of IPs.

a list of ports and their corresponding services can be obtained from $/etc/services$ for any unix system or from IANA `http://www.iana.org/assignments/port-numbers`.

What these four tables make clear is that there are *both* categories of targeted scanners—those that issue many scan attempts against specific sites, and those that issue as few attempts as possible. The latter make intuitive sense; the former, however, are more unusual; if a scanner is going to generate so many reports against disparate organizations, why would they not broadly scan other networks? One might chalk it up to coincidence, although the volume of the outliers (on the order of thousands of alerts) and scan length suggests that it is more than a coincidence.

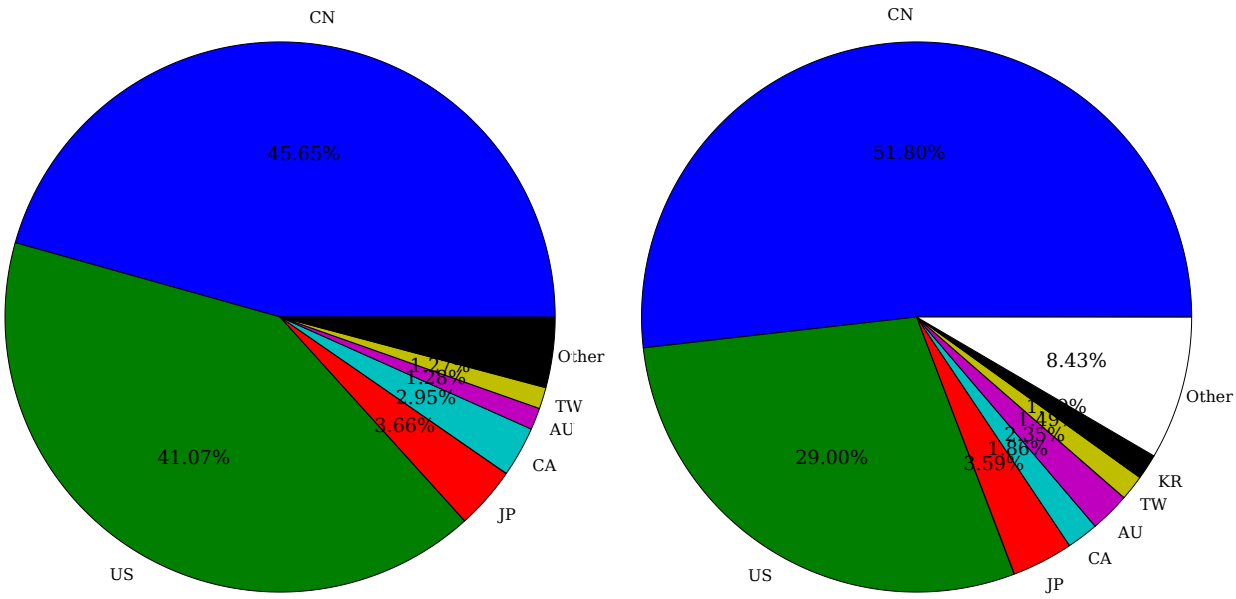

Fig. 18. Geographic distribution of 4-site scanners, by # of alerts and # of IPs.

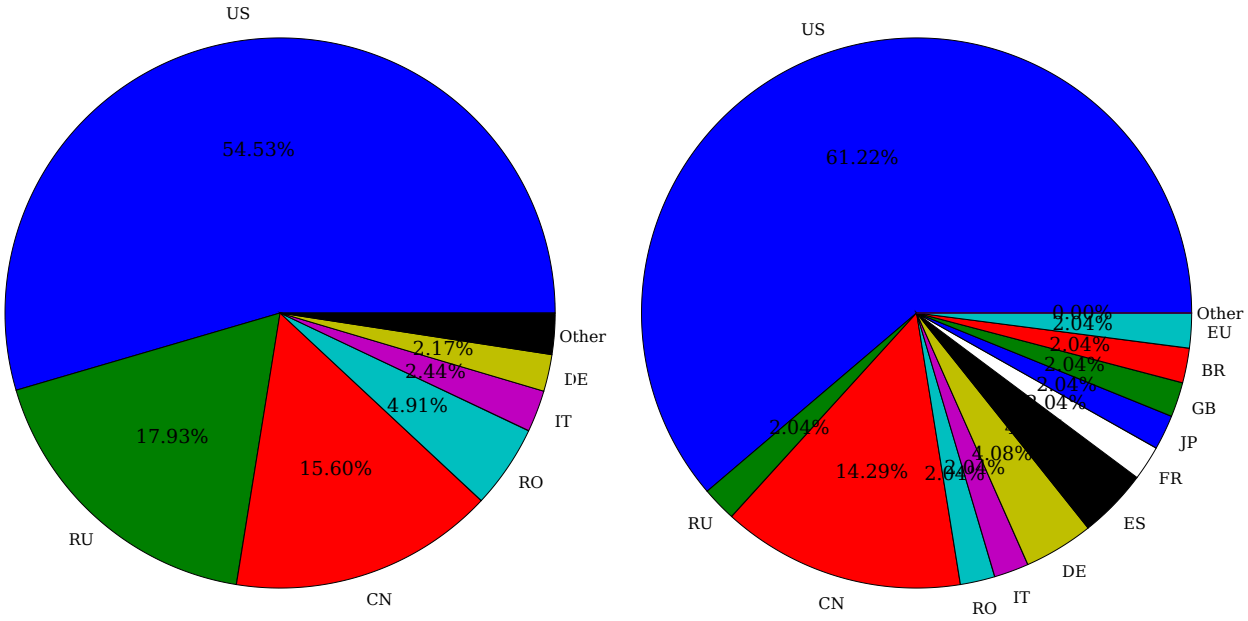

Fig. 19. Geographic distribution of 5-site scanners, by # of alerts and # of IPs.

Additionally, we can note that these scanners appear to be targeting different services across domains. For instance, academic sites seem to be more targeted for Messenger (port 1026) spam, while commercial sites seem to be targeted more for SQL vulnerabilities (port 1434); while both are discussed in greater detail in the next subsection, there may be both pragmatic and topological considerations behind these results.

In general, this form of analysis needs longer periods and broader data collection; ideally, such collection would enable analysis per *industry* or segments of industry, instead of the current rough-granular academic vs. commercial. Nevertheless, the results above show promise in this form of analysis.

# Min Sites	# /24s	Avg	StD	Max
1	208763	1.54	3.45	254
2	10939	1.40	1.91	115
3	2882	1.46	1.50	23
4	703	1.22	0.69	10
5	46	1.11	0.37	3

Table 8. Statistics on scanning subnets.

Source IP	Scan Length	# Alerts	Stealthiness	Top Ports
218.94.124.43	206.47	2913	1.633e-04	8080, 3128, 8000
202.63.188.2	19.87	1062	6.187e-04	25, 3389, 202
61.233.40.205	99.78	832	9.651e-05	1028, 1029, 1032
221.12.161.99	89.81	717	9.240e-05	1029, 1028, 1032
61.138.136.28	74.40	499	7.763e-05	4257, 1029, 1031
193.6.40.135	0.19	482	2.881e-02	22
202.103.86.66	184.99	473	2.959e-05	1026, 1027, 2
62.195.115.67	13.07	331	2.931e-04	1026
219.157.19.157	69.76	319	5.293e-05	1028, 1029, 1030
67.176.227.12	25.38	309	1.409e-04	1026

Table 9. Academic-only scanners, top 10 by # alerts.

Source IP	Scan Length	# Alerts	Stealthiness	Top Ports
194.204.0.1	288.15	5	2.008e-07	53, 54463, 54558
193.92.150.3	256.93	5	2.252e-07	53, 1118, 32877
64.15.205.211	278.54	6	2.493e-07	1086, 1118, 32877
200.23.242.197	277.62	6	2.501e-07	46319, 32877, 46796
213.150.135.213	259.05	6	2.681e-07	53, 1118, 32782
68.142.249.189	171.66	4	2.697e-07	80, 53, 443
213.140.2.12	286.52	7	2.828e-07	1118, 56156, 59916
203.116.1.78	294.10	8	3.148e-07	34269, 33020, 32840
194.85.82.254	229.77	7	3.526e-07	80, 443, 42
68.142.251.21	96.37	3	3.603e-07	80, 8080, 8060

Table 10. Academic-only scanners, top 10 by stealthiness.

Source IP	Scan Length	# Alerts	Stealthiness	Top Ports
61.241.93.47	2.38	732	3.555e-03	783, 2622, 762
195.7.3.100	168.75	566	3.882e-05	15118
221.202.84.227	368.21	537	1.688e-05	1434
61.175.218.186	171.19	467	3.157e-05	1434
62.210.4.23	9.69	368	4.396e-04	1434
61.139.54.94	171.68	345	2.326e-05	1434, 1433
62.233.215.129	17.04	340	2.309e-04	15118, 445
80.231.169.58	372.60	317	9.847e-06	1434
202.103.207.139	83.90	303	4.180e-05	1434
61.153.15.163	142.26	262	2.132e-05	1434

Table 11. Commercial-only scanners, top 10 by # alerts.

3.6 Targeted Services

The other specific targeting of interest is service—namely, what services are the broad scanners particularly looking at? And does this scan behavior differ significantly between narrow and broad scanners? Tables 21–30 show the top 15 targeted ports by 1- to 5-way scanning sources by both the number of IP and the frequency of alerts.

A few conclusions can be drawn from these figures. In particular, broad scanners frequently targeted different ports than narrow scanners. Most notable was port 1026; the most common service on that port is the Windows Messenger

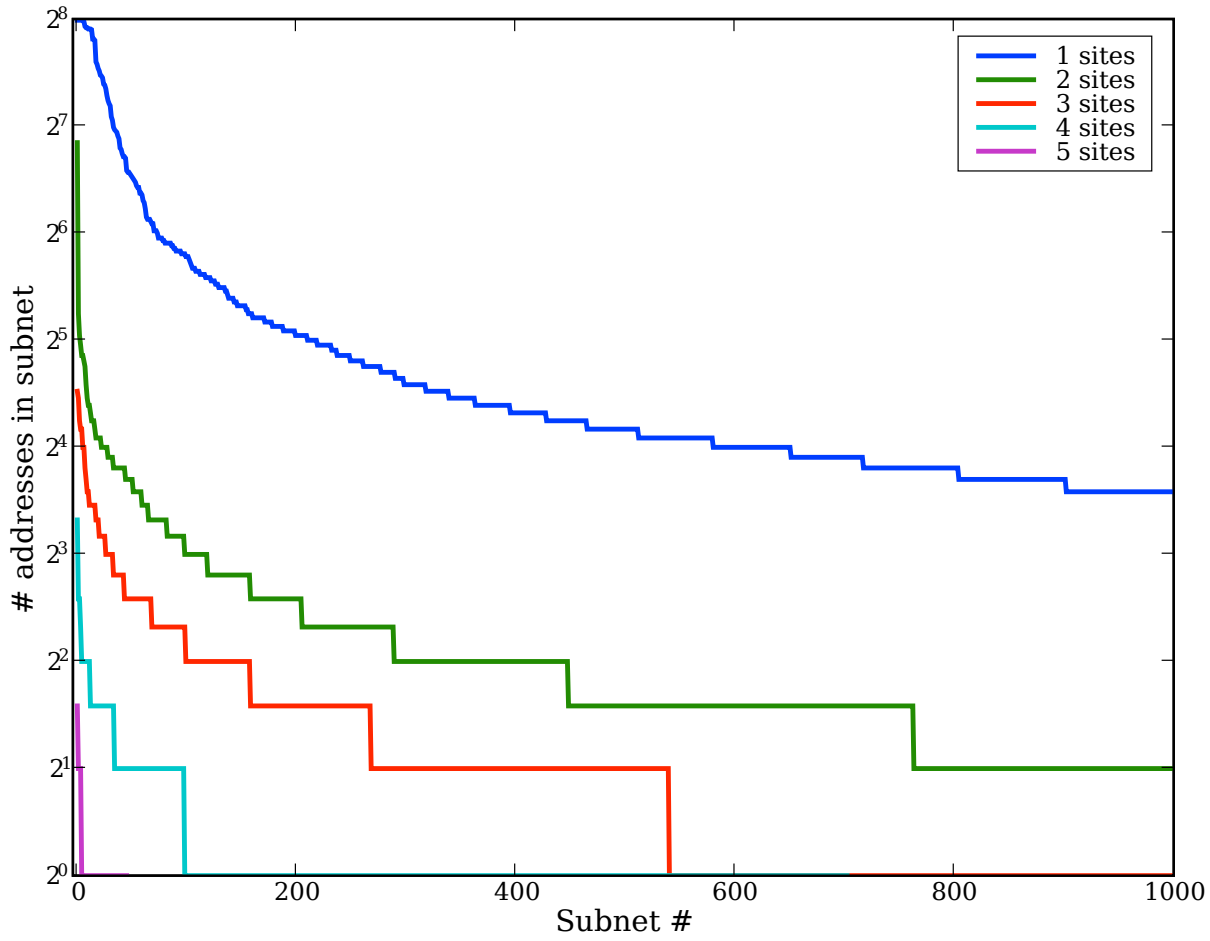

Fig. 20. Distribution of scanning subnet sizes by varying # min sites. The top line is 1 site, and the bottom/leftmost line is 5 sites.

Source IP	Scan Length	# Alerts	Stealthiness	Top Ports
69.157.8.5	286.74	3	1.211e-07	80
219.140.177.20	343.79	4	1.347e-07	1434
61.183.37.164	320.83	4	1.443e-07	43868, 139, 19547
80.188.58.18	306.44	4	1.511e-07	445, 135
221.15.233.166	301.49	4	1.536e-07	80
202.145.48.193	296.50	4	1.561e-07	139
200.149.32.170	290.44	4	1.594e-07	139, 135
218.95.64.229	287.86	4	1.608e-07	1434
69.157.174.218	281.53	4	1.644e-07	135
220.229.76.66	348.90	5	1.659e-07	139

Table 12. Commercial-only scanners, top 10 by stealthiness.

service, listening for UDP messages. During the time period of this data collection, Windows Messenger spam was still a major open target, and it appears many nodes were scanning for non-firewalled machines to deliver such spam. [4] What is interesting is that these alerts were far more noticeable when corroborated across sites, as opposed to individual site data. This may be due to the fact that unwanted or potentially malicious UDP traffic is far harder to detect via misuse analysis without generating too many false positives, and so many IDS analysis techniques reduce the weight of "suspicious UDP behavior" during their analysis and aggregation.

Port	# IPs	TotFreq
445	51537	999057
113	1087	967649
139	43079	790319
135	41054	753613
53	3320	635161
80	52716	452787
6881	819	309281
1025	60684	248797
1026	32966	244276
6346	1002	197356
1433	24683	185933
1434	9889	149863
25	1929	136517
137	6567	108274
33434	258	100873

Fig. 21. table
Top ports by frequency, 1+ site scans.

Port	# IPs	TotFreq
1026	2090	143753
33434	83	100067
53	533	98551
113	93	84649
1434	1380	83981
80	2685	65153
139	973	30470
1027	470	23372
137	303	15286
1024	578	14773
445	621	14169
3072	279	14120
1029	223	12583
1028	205	12370
25	282	11786

Fig. 22. table
Top ports by frequency, 2+ site scans.

Port	# IPs	TotFreq
1026	713	120073
33434	19	99128
1434	255	48687
1027	263	17332
80	431	16285
53	184	12578
1024	355	12334
3072	136	11851
113	38	11363
1029	107	8162
1028	103	8022
25	89	7721
1030	114	7113
1080	94	6288
135	414	5968

Fig. 23. table
Top ports by frequency, 3+ site scans.

Port	# IPs	TotFreq
1026	207	79425
1434	37	16786
1080	29	4742
1024	115	4453
3072	31	4196
135	98	3422
1029	33	3315
1028	30	3168
1030	30	2638
33437	21	2308
1027	101	2182
80	56	2131
33438	22	2015
33436	25	1792
33439	17	1596

Fig. 24. table
Top ports by frequency, 4+ site scans.

Port	# IPs	TotFreq
1026	11	6482
1080	4	3843
1029	6	2079
1028	5	1934
1030	5	1688
33436	9	880
135	7	842
33438	9	764
33437	8	744
33439	6	618
33440	4	376
33443	2	368
1434	1	352
80	4	346
22	2	335

Fig. 25. table
Top ports by frequency, 5+ site scans.

The same analysis can be applied to many of the high ephemeral ports, such as port 33435, 33436, etc. in tables 25 and 30; a quick investigation yields [2] that this is most likely Van Jacobsen-based traceroute packets, possibly due to newer load-balancers trying to redirect traffic to the closest datacenter. More data, and more sites, may prove to help glean more useful information in this regard. However, it is already clear that corroboration helps to identify several "chatty" ports that are not ordinarily detected by good misuse detectors, and analysis strategies should likely be reflected to include them.

One interesting change from 4-site to 5-site target results is the incidence reduction in port 1434 (MSSQL UDP). Further analysis suggests that one of the sites appeared to be blocking inbound port 1434 traffic, presumably as a preemptive firewalling strategy against Microsoft SQL-based worms. Ideally, a misuse sensor should be placed *outside* the firewall; barring that, given enough sites and analysis, it should be possible to discount certain data at certain sites due to firewalling or topology-specific considerations.

A more focused analysis of interest is to see which (IP, port) *tuples* have been observed across multiple sites. Tables 13 and 14 show the results for 4 and 5 sites, respectively, aggregated by scanner.

This form of analysis can, amongst other things, better help sites rank threats. If a site perceives certain services, as delineated by ports, as more vulnerable, they may choose to adopt more proactive stances against sources known to be broadly scanning those services across many sites, as opposed to ephemeral ports, possibly sign of a portscan or a

Port	# IPs	TotFreq
1025	60684	248797
80	52716	452787
445	51537	999057
139	43079	790319
135	41054	753613
1026	32966	244276
1433	24683	185933
6129	13236	62960
443	10339	59947
1434	9889	149863
2745	9068	83470
3127	8098	64363
137	6567	108274
5554	6151	23818
8080	5446	90736

Fig. 26. table
Top ports by # IPs, 1+ site scans.

Port	# IPs	TotFreq
80	2685	65153
1026	2090	143753
1434	1380	83981
135	1276	11556
1025	1175	11288
139	973	30470
34098	963	2519
22307	907	2034
54296	849	2320
23137	832	1977
1840	832	2467
26159	806	2163
14890	805	1718
1433	793	9753
49188	791	1971

Fig. 27. table
Top ports by # IPs, 2+ site scans.

Port	# IPs	TotFreq
1026	713	120073
22307	668	1601
23137	629	1613
34098	628	1763
14890	625	1399
26112	617	1460
50739	586	1497
6487	585	1241
54296	575	1658
26159	572	1598
1840	559	1709
14945	558	1406
54316	554	1313
20021	554	1276
11355	537	1292

Fig. 28. table
Top ports by # IPs, 3+ site scans.

Port	# IPs	TotFreq
1026	207	79425
26112	207	515
22307	206	515
6487	190	403
14890	188	424
34098	187	599
48148	182	399
60766	182	400
11355	180	479
5680	180	405
37948	178	395
4981	178	507
18183	175	383
5411	168	365
15201	167	375

Fig. 29. table
Top ports by # IPs, 4+ site scans.

Port	# IPs	TotFreq
1026	11	6482
33435	10	222
33436	9	880
33438	9	764
33437	8	744
135	7	842
137	7	258
1029	6	2079
33439	6	618
1028	5	1934
1030	5	1688
80	4	346
1027	4	177
1080	4	3843
33440	4	376

Fig. 30. table
Top ports by # IPs, 5+ site scans.

less-important protocol. The results in these tables suggest such a distribution; certain sources are very focused, e.g., 69.25.135.154, which is scanning primarily HTTP ports versus 64.94.45.30, which is likely computing traceroutes as discussed earlier.

References

1. Burton H. Bloom. Space/time trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.
2. Chase, Timothy. ISC SANS Intrusion Mailing List: UDP Traffic on Ports 33435-8, TCP on 2082 and 2745. `http://lists.sans.org/pipermail/intrusions/2004-August/008268.html`.
3. Jinyang Li, Jeremy Stribling, Thomer M. Gil, and Robert Morris. Comparing the Performance of Distributed Hash Tables Under Churn. In *International Workshop on Peer-to-Peer Systems (IPTPS)*, San Diego, CA, 2004.
4. LURHQ Threat Intelligence Group. Windows Messenger Popup Spam on UDP Port 1026. `http://www.lurhq.com/popup_spam.html`.
5. Bill McCarty. Botnets: Big and Bigger. *IEEE Security and Privacy*, 1(4):87–90, 2003.
6. Janak J. Parekh, Ke Wang, and Salvatore J. Stolfo. Privacy-Preserving Payload-Based Correlation for Accurate Malicious Traffic Detection. Technical report, Columbia University Dept. of CS, 2006. `http://mice.cs.columbia.edu/getTechreport.php?techreportID=409`.

Source IP	# Ports	Top Ports
211.154.222.56	81	1917, 1803, 1911, 1263, 1352
209.208.0.15	66	1080, 40934, 41457, 1813, 1978
218.30.70.56	49	1393, 1151, 1928, 1093, 1295
61.145.127.92	45	1614, 1674, 1450, 1286, 1087
60.31.184.7	29	1865, 1682, 1660, 1641, 1563
221.174.17.252	28	1639, 1499, 1834, 1642, 1577
69.25.135.154	7	8000, 81, 80, 8080, 8081
82.96.96.3	7	3128, 3802, 3777, 6588, 8080
65.223.84.131	6	2301, 8000, 80, 8080, 3124
199.181.135.4	6	33438, 33440, 33439, 33437, 33436
66.219.100.118	5	1080, 3128, 6588, 8080, 80
161.170.254.232	5	33438, 33436, 33441, 33444, 33435
166.91.254.4	5	2968, 2967, 2970, 2969, 8081
216.183.96.100	5	33439, 33438, 33437, 33436, 33435
61.137.117.208	4	1027, 1026, 1029, 1028

Table 13. Most popular (IP, port) tuples by source IP seen at 4 sites.

Source IP	# Ports	Top Ports
209.208.0.15	65	1080, 40934, 41457, 1813, 1978
216.183.96.100	4	33439, 33438, 33437, 33436
69.20.1.77	2	33440, 33438
221.12.161.109	2	1026, 1027
24.164.180.228	1	1026
61.141.32.80	1	80
64.94.45.30	1	33443
66.150.223.54	1	33443
66.151.55.10	1	33440
66.151.55.30	1	33439
66.179.168.100	1	33437
69.25.27.10	1	33440
69.40.165.231	1	1026
69.133.97.207	1	1026
81.74.106.18	1	1026

Table 14. Most popular (IP, port) tuples by source IP seen at 5 sites.

7. Honeynet Project and Research Alliance. Know your Enemy: Tracking Botnets, 3/13/05 2005. `http://www.honeynet.org/papers/bots/`.
8. Seth Robertson, Eric V. Siegel, Matt Miller, and Salvatore J. Stolfo. Surveillance Detection in High Bandwidth Environments. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, 2003.
9. Lambert Schaelicke, Matthew R. Geiger, and Curt J. Freeland. Improving the Database Logging Performance of the Snort Network Intrusion Detection Sensor. Technical report, University of Notre Dame, 2002.
10. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, San Diego, 2001. ACM.
11. Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation. In *Symposium on Recent Advances in Intrusion Detection*, Seattle, WA, 2005.
12. Ke Wang, Janak J. Parekh, and Salvatore J. Stolfo. Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In *Symposium on Recent Advances in Intrusion Detection*, Hamburg, Germany, 2006.
13. Ke Wang and Salvatore J. Stolfo. Anomalous Payload-based Network Intrusion Detection. In *Symposium on Recent Advances in Intrusion Detection*, Sophia Antipolis, France, 2004.